FPSE Project Design: Strands

Tam, Mia, Emma, David

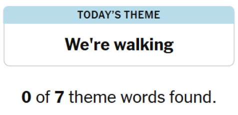
Overview of the Purpose of the Project

- The purpose of this project is to recreate "Strands", a word search game by the New York Times. Strands feature a 6x8 letter grid where players find words linked by a common theme. A unique "spangram," a challenge word connecting two opposite board edges, provides a major hint to the theme. Themed words highlight in blue, while the spangram appears in yellow. Each puzzle must be fully solved with no overlapping words.

A Complete Mock Use of the Application

GUI

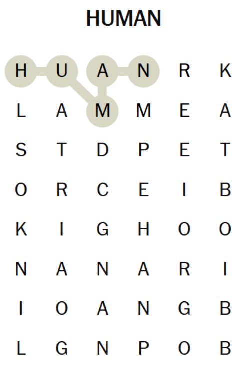
- We would like to emulate the design of the existing NYT game with a few modifications

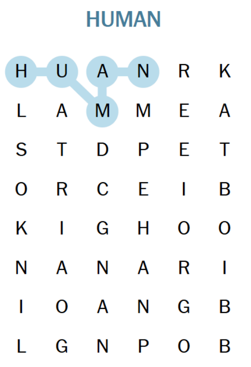


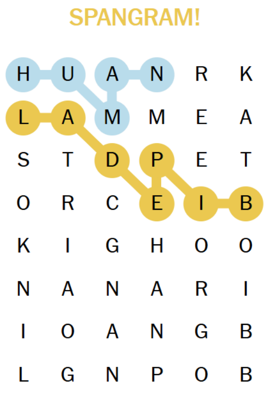
TODAY'S THEME

We're walking

0 of **7** theme words found.







Usage feature 1: A well-formed grid is generated and displayed

- A well-formed grid will have the following requirements
  - The grid is 6 x 8 with no empty squares
  - The grid contains all words that were used to initialize the puzzle
  - Words do cross over themselves or across other words
  - The grid contains a "Spangram" which can be horizontal or vertical and must extend from one edge to the other
- Other considerations:
  - The puzzle must have an element of randomness - there should be variety in the layout of the words and no two puzzles should have an identical layout

    - The puzzle must be reasonably complex - meaning that the words cannot be simply displayed in easily decipherable rows/columns

Usage feature 2: Users can select words on the grid

- Users should be able to select words on the grid to make their guess by dragging their cursor (depending on how our front-end is designed, we may find it easier for the user to click to select words and have a "submit guess" button)
- The user's selection must not cross over itself or across the path of any other words
- The user should be able to "undo" their selection either by scrolling over it or clicking on a selected letter (depending on our implementation)

Usage feature 3: Identified words must have feedback associated with them

- Correct words will be highlighted in blue after the user has identified them
- The Spangram will be highlighted in yellow after the user identifies it
- Once a user identifies a word, the counter that displays the total words found should increment
- Incorrectly selected words must provide some form of visual feedback (i.e. "that word is not in our dictionary" or "that is an invalid guess") and then reset the board to its previous state

Usage feature 4: Completing the game presents a "victory" state

- Once the user identifies every word, they should be alerted to their success with a victory message

Potential usage feature: User can create their own Strands puzzle

- Users can submit a text file or input words into a textbox to generate their own Strands puzzle

Potential Built-In OCaml Libraries/ Modules For Implementation

- OCaml library : Array
- Module Stdlib - OCaml library
- OCaml library : List
- OCaml library : Random

See the video in our README or linked here

- Rescript React for front-end
    - OCaml with a different looking syntax that compiles to JavaScript and has bindings for React; can run in the browser
- opium 0.19.0 · OCaml Package
    - Opium as the web framework (binds functions to http routes) - (Note from Mia: I set this up and it is working for now, but I might end up switching this to Dream)

Implementation Plan:

Algorithm

- Set Cover Problem - we need to ensure that every position of our grid is covered by a letter from a word, with the spangram word spanning from one side of the grid to the other.

- https://customstrandsnyt.com/about

Frontend Implementation
We will use ReScript React for our frontend. ReScript is built on top of OCaml and we can build a React application through its ReScript React bindings. This will be important as our gameplay depends on a strong GUI.

Components:
- Game - main page that will contain gameplay elements such as the Words Found counter and grid
- Grid - will contain the Letter components. The grid will be initialized by our backend and rendered in the grid component.
- Letter - will make up the individual letters of the grid and will handle the various displays depending on the state of the word (selected, found, spangram, etc.)

Backend Implementation - Game Setup

Collection of Themed Words from our Text File(s)
- Word Dictionary Formatting ⟶ (int * string list)
    - Example shown in words.txt
- Constraints: All words presented have to be length >= 4
- Can have multiple text files for different "themes"
- Optional / Room for Improvement:
    - Have the user make their own themed words

[Completed words.txt example, need to add more text files later on for more variety]

utils
- We defined a utils file that will help us generate a sample of words for the grid
- Based on the formatting of our text file(s), we defined
    - type word_dict ⟶ serves as a dictionary of words groups by length
- let read_words filename ⟶ reading from the text file
- let select_words word_dict total_chars
    - this function will accumulate a list of selected words until we reach **48 total letters** from the list

[these util functions should be done soon after THURS OH 11/14/24]

grid
- Grid Creation
    - Our grid will be 6 x 8 2D array, initialized each cell (row, col) with empty chars ‘ ‘
- We defined a Grid module
    - type t ⟶ underlying type represents the grid itself (char array array)
    - type position ⟶ to represent the cells on the grid (int * int)

        - **potential type for tracking a word's path **
- Selecting and Placing the Spangram
    - Select a spangram
        - Pick from the list of selected words (from utils)
            - It must stretch from one side of the grid to the other side, either horizontally or vertically, hence its length will >= 6
            - If the spangram is placed horizontally, its length should be 6 (to fill one row entirely). If placed vertically, it should be 8
    - Placing the spangram on the grid (G) ⟶
        - select_spangram and place_spangram functions (other potentially more functions) will implement this algorithm
            - Randomly decide if we want to place it horizontal or vertical
            - Run BFS on the grid G from the own border to represent min edges required to each other side. Store in a grid M
            - On each iteration, place a letter on G and choose a neighbor N if remaining characters in S <= M[N]. Then update M by running BFS again
            - Let W be the set of words. After placing S in G, run BFS on either side of S to the available sports G1, G2.
            - Run subset sum with W as the array, and |G1| being the target
            - If subset sum returns false, reset G, S and restart

- Populating Remaining Themed Words (W) into the Grid without Overlaps
    - Val populate_grid grid words
        - Partition W into W1, W2 based on subset sum
        - Track the path of each placed word

- Print out the Board
    - let print_board grid ⟶ print the generated grid in a readable format

[finalize mli file by the end of this week]
[Looking to finish the above functions by Nov Mon 25]

game

- We defined a type word = string and a type position
    - To identity and track the word the player has selected
- Marking a word as found by the player
- Checking if the player's selected word is a spangram
- Checking if the player's selected word is one the themed list of words
- Checking if the player's selected word has more than 3 characters
- Printing and updating the list of words the player has found so far

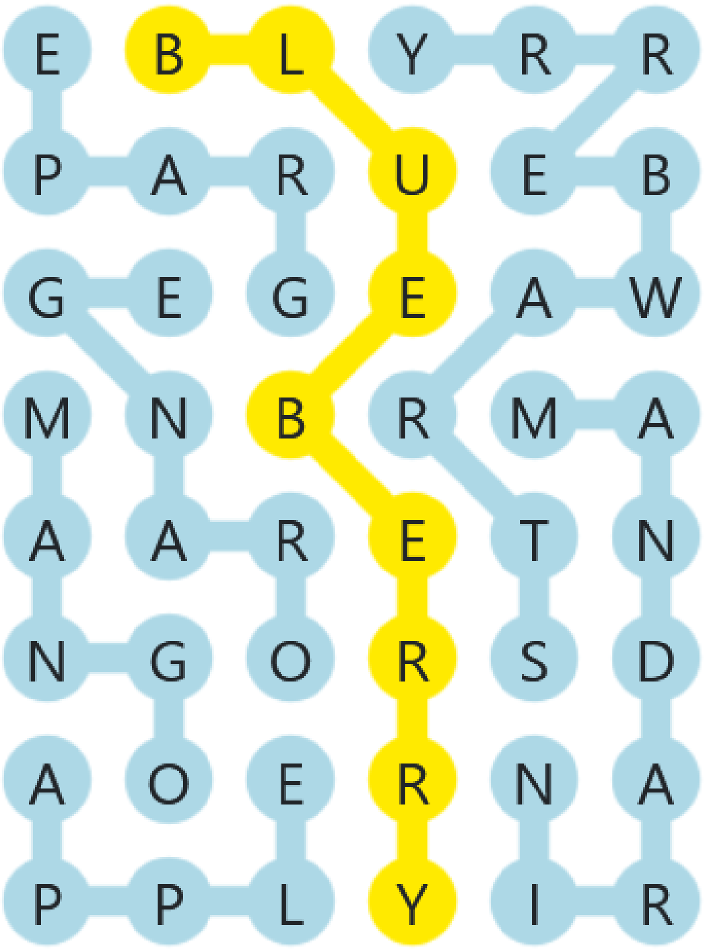
[finalize mli file by the end of this week]
[Looking to finish the above functions by Dec Mon 2]

Testing

- We will test functions in each of our library files using the OUnit2 testing library
- We could potentially use Base QuickCheck to test invariant conditions/valid words to use in our implementation
- We will also use Bisect to test our coverage, aiming for a 95% coverage
- We will define test suites for different specification / finite window of behaviors

- A stretch for testing could be front-end testing but I think that would be beyond the scope of this assignment. However, we will user test the functionality critically.

The following image displays the testing sample.txt we will use to test our functions found in strand_set.txt:


E B L Y R R
P A R U E B
G E G E A W
M N B R M A
A A R E T N
N G O R S D
A O E R N A
P P L Y I R